Čížek, Pavel; Čížková, Lenka

Working Paper

Numerical Linear Algebra

Papers, No. 2004,23

Provided in Cooperation with:
CASE - Center for Applied Statistics and Economics, Humboldt University Berlin

Suggested Citation: Čížek, Pavel; Čížková, Lenka (2004) : Numerical Linear Algebra, Papers, No. 2004,23, Humboldt-Universität zu Berlin, Center for Applied Statistics and Economics (CASE), Berlin

This Version is available at:
https://hdl.handle.net/10419/22197

Numerical Linear Algebra

Lenka Čížková[1] and Pavel Čížek[2]

[1] Faculty of Nuclear Sciences and Physical Engineering, Czech Technical University in Prague, Břehová 7, 115 19 Praha 1, The Czech Republic
lenka_cizkova@web.de
[2] Tilburg University, Department of Econometrics & Operations Research Room B 516, P.O. Box 90153, 5000 LE Tilburg, The Netherlands
P.Cizek@uvt.nl

Many methods of computational statistics lead to matrix-algebra or numerical-mathematics problems. For example, the least squares method in linear regression reduces to solving a system of linear equations, see Chapter **??**. The principal components method is based on finding eigenvalues and eigenvectors of a matrix, see Chapter **??**. Nonlinear optimization methods such as Newton's method often employ the inversion of a Hessian matrix. In all these cases, we need numerical linear algebra.

Usually, one has a data matrix $\mathbf{X}$ of (explanatory) variables, and in the case of regression, a data vector $\mathbf{y}$ for dependent variable. Then the matrix defining a system of equations, being inverted or decomposed typically corresponds to $\mathbf{X}$ or $\mathbf{X}^\top\mathbf{X}$. We refer to the matrix being analyzed as $\mathbf{A} = \{A_{ij}\}_{i=1,j=1}^{m,n} \in \mathbb{R}^{m\times n}$ and to its columns as $\mathbf{A}_k = \{A_{ik}\}_{i=1}^{m}, k = 1,\ldots,n$. In the case of linear equations, $\mathbf{b} = \{b_i\}_{i=1}^{n} \in \mathbb{R}^n$ represents the right-hand side throughout this chapter. Further, the eigenvalues and singular values of $\mathbf{A}$ are denoted by λ_i and σ_i, respectively, and the corresponding eigenvectors $\mathbf{g}_i$, $i = 1,\ldots,n$. Finally, we denote the $n \times n$ identity and zero matrices by $\mathbf{I_n}$ and $\mathbf{0_n}$, respectively.

In this chapter, we first study various matrix decompositions (Section 1), which facilitate numerically stable algorithms for solving systems of linear equations and matrix inversions. Next, we discuss specific direct and iterative methods for solving linear systems (Sections 2 and 3). Further, we concentrate on finding eigenvalues and eigenvectors of a matrix in Section 4. Finally, we provide an overview of numerical methods for large problems with sparse matrices (Section 5).

Let us note that most of the mentioned methods work under specific conditions given in existence theorems, which we state without proofs. Unless said otherwise, the proofs can be found in Harville (1997), for instance. Moreover, implementations of the algorithms described here can be found, for example, in Anderson et al. (1999) and Press et al. (1992).

1 Matrix Decompositions

This section covers relevant matrix decompositions and basic numerical methods. Decompositions provide a numerically stable way to solve a system of linear equations, as shown already in Wampler (1970), and to invert a matrix. Additionally, they provide an important tool for analyzing the numerical stability of a system.

Some of most frequently used decompositions are the Cholesky, QR, LU, and SVD decompositions. We start with the Cholesky and LU decompositions, which work only with positive definite and nonsingular diagonally dominant square matrices, respectively (Sections 1.1 and 1.2). Later, we explore more general and in statistics more widely used QR and SVD decompositions, which can be applied to any matrix (Sections 1.3 and 1.4). Finally, we briefly describe the use of decompositions for matrix inversion, although one rarely needs to invert a matrix (Section 1.5). Monographs Gentle (1998), Harville (1997), Higham (2002) and Stewart (1998) include extensive discussions of matrix decompositions.

All mentioned decompositions allow us to transform a general system of linear equations to a system with an upper triangular, a diagonal, or a lower triangular coefficient matrix: $\mathbf{U}\mathbf{x} = \mathbf{b}, \mathbf{D}\mathbf{x} = \mathbf{b}$, or $\mathbf{L}\mathbf{x} = \mathbf{b}$, respectively. Such systems are easy to solve with a very high accuracy by back substitution, see Higham (1989). Assuming the respective coefficient matrix $\mathbf{A}$ has a full rank, one can find a solution of $\mathbf{U}\mathbf{x} = \mathbf{b}$, where $\mathbf{U} = \{U_{ij}\}_{i=1,j=1}^{n}$, by evaluating

$$x_i = U_{ii}^{-1} \left(b_i - \sum_{j=i+1}^{n} U_{ij} x_j \right) \tag{1}$$

for $i = n, \ldots, 1$. Analogously for $\mathbf{L}\mathbf{x} = \mathbf{b}$, where $\mathbf{L} = \{L_{ij}\}_{i=1,j=1}^{n}$, one evaluates for $i = 1, \ldots, n$

$$x_i = L_{ii}^{-1} \left(b_i - \sum_{j=1}^{i-1} L_{ij} x_j \right). \tag{2}$$

For discussion of systems of equations that do not have a full rank, see for example Higham (2002).

1.1 Cholesky Decomposition

The Cholesky factorization, first published by Benoit (1924), was originally developed to solve least squares problems in geodesy and topography. This factorization, in statistics also referred to as "square root method," is a triangular decomposition. Providing matrix $\mathbf{A}$ is positive definite, the Cholesky decomposition finds a triangular matrix $\mathbf{U}$ that multiplied by its own transpose leads back to matrix $\mathbf{A}$. That is, $\mathbf{U}$ can be thought of as a square root of $\mathbf{A}$.

Theorem 1. *Let matrix* $\mathbf{A} \in \mathbb{R}^{n\times n}$ *be symmetric and positive definite. Then there exists a unique upper triangular matrix* $\mathbf{U}$ *with positive diagonal elements such that* $\mathbf{A} = \mathbf{U}^\top\mathbf{U}$.

The matrix $\mathbf{U}$ is called the Cholesky factor of $\mathbf{A}$ and the relation $\mathbf{A} = \mathbf{U}^\top\mathbf{U}$ is called the Cholesky factorization.

Obviously, decomposing a system $\mathbf{Ax} = \mathbf{b}$ to $\mathbf{U}^\top\mathbf{Ux} = \mathbf{b}$ allows us to solve two triangular systems: $\mathbf{U}^\top\mathbf{z} = \mathbf{b}$ for $\mathbf{z}$ and then $\mathbf{Ux} = \mathbf{z}$ for $\mathbf{x}$. This is similar to the original Gauss approach for solving a positive definite system of normal equations $\mathbf{X}^\top\mathbf{Xx} = \mathbf{X}^\top\mathbf{b}$. Gauss solved the normal equations by a symmetry-preserving elimination and used the back substitution to solve for $\mathbf{x}$.

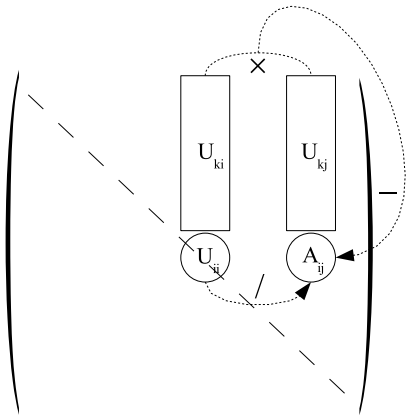

Fig. 1. Rowwise Cholesky algorithm

Let us now describe the algorithm for finding the Cholesky decomposition, which is illustrated on Figure 1. One of the interesting features of the algorithm is that in the ith iteration we obtain the Cholesky decomposition of the ith leading principal minor of $\mathbf{A}$, $\{A_{kl}\}_{k=1,l=1}^{i,i}$.

Algorithm 1

```
for i=1 to n
    U_ii = ( A_ii − Σ_{k=1}^{i−1} U_ki^2 )^{1/2}
    for j=i+1 to n
        U_ij = ( A_ij − Σ_{k=1}^{i−1} U_ki U_kj ) / U_ii
    end
end
```

The Cholesky decomposition described in Algorithm 1 is a numerically stable procedure, see Martin et al. (1965) and Meinguet (1983), which can be

at the same time implemented in a very efficient way. Computed values U_{ij} can be stored in place of original A_{ij}, and thus, no extra memory is needed. Moreover, let us note that while Algorithm 1 describes the rowwise decomposition ($\mathbf{U}$ is computed row by row), there are also a columnwise version and a version with diagonal pivoting, which is also applicable to semidefinite matrices. Finally, there are also modifications of the algorithm, such as blockwise decomposition, that are suitable for very large problems and parallelization; see Björck (1996), Gallivan et al. (1990) and Nool (1995).

1.2 LU Decomposition

The LU decomposition is another method reducing a square matrix $\mathbf{A}$ to a product of two triangular matrices (lower triangular $\mathbf{L}$ and upper triangular $\mathbf{U}$). Contrary to the Cholesky decomposition, it does not require a positive definite matrix $\mathbf{A}$, but there is no guarantee that $\mathbf{L} = \mathbf{U}^\top$.

Theorem 2. *Let the matrix* $\mathbf{A} \in \mathbb{R}^{n \times n}$ *satisfy following conditions:*

$$A_{11} \neq 0, \quad \det \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} \neq 0, \quad \det \begin{pmatrix} A_{11} & A_{12} & A_{13} \\ A_{21} & A_{22} & A_{23} \\ A_{31} & A_{32} & A_{33} \end{pmatrix} \neq 0, \quad \ldots, \quad \det \mathbf{A} \neq 0.$$

Then there exists a unique lower triangular matrix $\mathbf{L}$ *with ones on a diagonal, a unique upper triangular matrix* $\mathbf{U}$ *with ones on a diagonal and a unique diagonal matrix* $\mathbf{D}$ *such that* $\mathbf{A} = \mathbf{LDU}$.

Note that for any nonsingular matrix $\mathbf{A}$ there is always a row permutation $\mathbf{P}$ such that the permuted matrix $\mathbf{PA}$ satisfies the assumptions of Theorem 2. Further, a more frequently used version of this theorem factorizes $\mathbf{A}$ to a lower triangular matrix $\mathbf{L}' = \mathbf{LD}$ and an upper triangular matrix $\mathbf{U}' = \mathbf{U}$. Finally, Zou (1991) gave alternative conditions for the existence of the LU decomposition: $\mathbf{A} \in \mathbb{R}^{n \times n}$ is nonsingular and $\mathbf{A}^\top$ is diagonally dominant (i.e., $|A_{ii}| \geq \sum_{i=1, i \neq j}^{n} |A_{ij}|$ for $j = 1, \ldots, n$).

Similarly to the Cholesky decomposition, the LU decomposition reduces solving a system of linear equations $\mathbf{Ax} = \mathbf{LUx} = \mathbf{b}$ to solving two triangular systems: $\mathbf{Lz} = \mathbf{b}$, where $\mathbf{z} = \mathbf{Ux}$, and $\mathbf{Ux} = \mathbf{z}$.

Finding the LU decomposition of $\mathbf{A}$ is described in Algorithm 2. Since it is equivalent to solving a system of linear equations by the Gauss elimination, which searches just for $\mathbf{U}$ and ignores $\mathbf{L}$, we refer a reader to Section 2, where its advantages (e.g., easy implementation, speed) and disadvantages (e.g., numerical instability without pivoting) are discussed.

Algorithm 2

$\mathbf{L} = \mathbf{0_n}, \mathbf{U} = \mathbf{I_n}$

```
for i = 1 to n
    for j = i to n
```

$L_{ji} = A_{ji} - \sum_{k=1}^{i-1} L_{jk}U_{ki}$
    end
    for j = i + 1 to n
$U_{ij} = \left(A_{ij} - \sum_{k=1}^{i-1} L_{ik}U_{kj}\right)/L_{ii}$
    end
end

Finally, note that there are also generalizations of LU to non-square and singular matrices, such as rank revealing LU factorization; see Pan (2000) and Miranian and Gu (2003).

1.3 QR Decomposition

One of the most important matrix transformations is the QR decomposition. It splits a general matrix $\mathbf{A}$ to an orthonormal matrix $\mathbf{Q}$, that is, a matrix with columns orthogonal to each other and its Euclidian norm equal to 1, and to an upper triangular matrix $\mathbf{R}$. Thus, a suitably chosen orthogonal matrix $\mathbf{Q}$ will triangularize the given matrix $\mathbf{A}$.

Theorem 3. *Let matrix $\mathbf{A} \in \mathbb{R}^{m\times n}$ with $m \geq n$. Then there exist an orthonormal matrix $\mathbf{Q} \in \mathbb{R}^{m\times m}$ and an upper triangular matrix $\mathbf{R} \in \mathbb{R}^{n\times n}$ with nonnegative diagonal elements such that*

$$\mathbf{A} = \mathbf{Q}\begin{pmatrix}\mathbf{R}\\ \mathbf{0}\end{pmatrix}$$

(the QR decomposition of the matrix $\mathbf{A}$).

If $\mathbf{A}$ is a nonsingular square matrix, an even slightly stronger result can be obtained: uniqueness of the QR decomposition.

Theorem 4. *Let matrix $\mathbf{A} \in \mathbb{R}^{n\times n}$ be nonsingular. Then there exist a unique orthonormal matrix $\mathbf{Q} \in \mathbb{R}^{n\times n}$ and a unique upper triangular matrix $\mathbf{R} \in \mathbb{R}^{n\times n}$ with positive diagonal elements such that $\mathbf{A} = \mathbf{QR}$.*

The use of the QR decomposition for solving a system of equations $\mathbf{Ax} = \mathbf{QRx} = \mathbf{b}$ consists in multiplying the whole system by the orthonormal matrix $\mathbf{Q}^\top$, $\mathbf{Q}^\top\mathbf{Q} = \mathbf{I}$, and then solving the remaining upper triangular system $\mathbf{Rx} = \mathbf{Q}^\top\mathbf{b}$. This method guarantees numerical stability by minimizing errors caused by machine roundoffs (see the end of this section for details).

The QR decomposition is usually constructed by finding one orthonormal vector (one column of $\mathbf{Q}$) after another. This can be achieved using the so-called elementary orthogonal transformations such as Householder reflections, Householder (1958), or Givens rotations, Givens (1958), that are described in the following subsections. These transformations are related to the solution of the following standard task.

Problem 1. Given a vector $\mathbf{x} \in \mathbb{R}^m$, $\mathbf{x} \neq 0$, find an orthogonal matrix $\mathbf{M} \in \mathbb{R}^{m \times m}$ such that $\mathbf{M}^\top \mathbf{x} = \|\mathbf{x}\|_2 \cdot \mathbf{e}_1$, where $\mathbf{e}_1 = (1, 0, \ldots, 0)^\top$ denotes the first unit vector.

In the rest of this section, we will first discuss how Householder reflections and Givens rotations can be used for solving Problem 1. Next, using these elementary results, we show how one can construct the QR decomposition. Finally, we briefly mention the Gram-Schmidt orthogonalization method, which also provides a way to find the QR decomposition.

Householder Reflections

The QR decomposition using Householder reflections (HR) was developed by Golub (1965). Householder reflection (or Householder transformation) is a matrix $\mathbf{P}$,

$$\mathbf{P} = \mathbf{I} - \frac{1}{c}\mathbf{u}\mathbf{u}^\top, \quad c = \frac{1}{2}\mathbf{u}^\top \mathbf{u}, \tag{3}$$

where $\mathbf{u}$ is a Householder vector. By definition, the matrix $\mathbf{P}$ is orthonormal and symmetric. Moreover, for any $\mathbf{x} \in \mathbb{R}^m$, it holds that

$$\mathbf{P}\mathbf{x} = \mathbf{x} - \frac{1}{c}(\mathbf{u}^\top \mathbf{x})\mathbf{u}.$$

Therefore, to apply HR one does not need to explicitly compute the matrix $\mathbf{P}$ itself. Additionally, it holds $\mathbf{P}\mathbf{u} = -\mathbf{u}$ and $\mathbf{P}\mathbf{x} \in \text{span}\{\mathbf{x}, \mathbf{u}\}$. This means that HR reflects a vector $\mathbf{x}$ with respect to the hyperplane with normal vector $\mathbf{u}$ (hence the name Householder reflection).

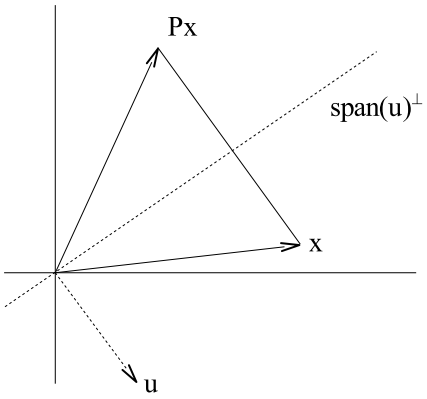

Fig. 2. Reflection with respect to the hyperplane with a normal vector $\mathbf{u}$.

To solve Problem 1 using some HR, we search for a vector $\mathbf{u}$ such that $\mathbf{x}$ will be reflected to the x-axis. This holds for the following choice of $\mathbf{u}$:

$$\mathbf{u} = \mathbf{x} + s_1 \|\mathbf{x}\|_2 \cdot \mathbf{e}_1, \qquad s_1 = 2I(x_1 \geq 0) - 1, \tag{4}$$

where $x_1 = \mathbf{x}^\top \mathbf{e}_1$ denotes the first element of the vector $\mathbf{x}$ and $I(\cdot)$ represents an indicator. For this reflection, it holds that c from (3) equals $\|x\|_2(\|x\|_2 + |x_1|)$ and $\mathbf{P}\mathbf{x} = -s_1\|\mathbf{x}\|_2 \cdot \mathbf{e}_1$ as one can verify by substituting (4) into (3).

Givens Rotations

A Givens rotation (GR) in m dimensions (or Givens transformation) is defined by an orthonormal matrix $\mathbf{R}_{ij}(\alpha) \in \mathbb{R}^{m \times m}$,

$$\mathbf{R}_{ij}(\alpha) = \begin{pmatrix} 1 & 0 & \cdots & \cdots & \cdots & \cdots & 0 \\ 0 & \ddots & & & & & \vdots \\ \vdots & & c & & s & & \vdots \\ \vdots & & & \ddots & & & \vdots \\ \vdots & & -s & & c & & \vdots \\ \vdots & & & & & \ddots & 0 \\ 0 & \cdots & \cdots & \cdots & \cdots & 0 & 1 \end{pmatrix} \begin{matrix} \\ \\ i \\ \\ j \\ \\ \end{matrix} \tag{5}$$

where $c = \cos\alpha$ and $s = \sin\alpha$ for $\alpha \in \mathbb{R}$ and $1 \leq i < j \leq n$. Thus, the rotation $\mathbf{R}_{ij}(\alpha)$ represents a plane rotation in the space spanned by the unit vectors $\mathbf{e}_i$ and $\mathbf{e}_j$ by an angle α. In two dimensions, rotation $\mathbf{R}_{12}(\alpha)$,

$$\mathbf{R}_{12}(\alpha) = \begin{pmatrix} c & s \\ -s & c \end{pmatrix}, \quad c = \cos\alpha, \quad s = \sin\alpha$$

represents a clockwise rotation by an angle α; see Figure 3.

Now, let us have a look at how GRs can be used for solving Problem 1. A GR of a vector $\mathbf{x} = (x_1, \ldots, x_m)^\top \in \mathbb{R}^m$ by an angle α results in $\mathbf{R}_{ij}(\alpha)\mathbf{x} = \mathbf{y} = (y_1, \ldots, y_m)^\top$ such that

$$y_k = \begin{cases} x_k & \text{for } k \neq i, j, \\ cx_i + sx_j & \text{for } k = i, \\ -sx_i + cx_j & \text{for } k = j. \end{cases}$$

For a vector $\mathbf{x}$ with nonzero elements x_i or x_j, setting $d = (x_i^2 + x_j^2)^{1/2}$, $c = x_i/d$, $s = x_j/d$ leads to

$$\begin{pmatrix} c & s \\ -s & c \end{pmatrix} \begin{pmatrix} x_i \\ x_j \end{pmatrix} = \begin{pmatrix} d \\ 0 \end{pmatrix}.$$

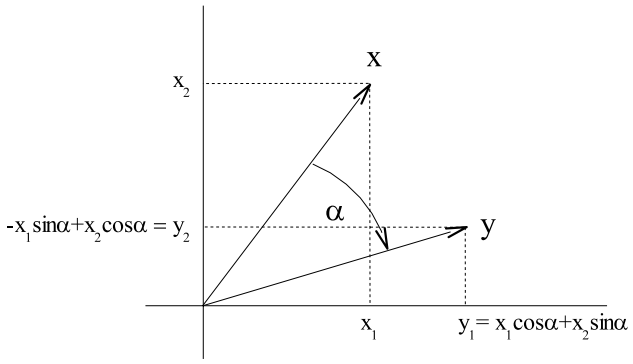

Fig. 3. Rotation of $\mathbf{x}$ in a plane by an angle α.

Thus, using GR with this specific choice of c and s (referred further as $\mathbf{R}^0_{ij}$) implies that the jth component of the vector $\mathbf{x}$ vanishes. Similarly to HRs, it is not necessary to explicitly construct the whole matrix $\mathbf{P}$ to transform $\mathbf{x}$ since the rotation is fully described by only two numbers: c and s. This elementary rotation $\mathbf{R}^0_{ij}$ does not however constitute a solution to Problem 1 yet: we need to combine more of them.

The next step employs a simple fact that the pre- or postmultiplication of a vector $\mathbf{x}$ or a matrix $\mathbf{A}$ by any GR $\mathbf{R}_{ij}(\alpha)$ affects only the ith and jth rows and columns, respectively. Hence, one can combine several rotations without one rotation spoiling the result of another rotation. (Consequently, GRs are more flexible than HRs). Two typical ways how GRs are used for solving Problem 1 mentioned in Section 1.3 follow.

1. $\mathbf{R}^0_{1n}\mathbf{R}^0_{1,n-1}\dots\mathbf{R}^0_{13}\mathbf{R}^0_{12}\mathbf{x} = d\mathbf{e}_1$. Here the kth component of the vector $\mathbf{x}$ vanishes after the Givens rotation $\mathbf{R}^0_{1k}$. The previously zeroed elements $x_2, \dots, x_{k-1}$ are not changed because rotation $\mathbf{R}_{1k}$ affects only the first and kth component.
2. $\mathbf{R}^0_{12}\mathbf{R}^0_{23}\dots\mathbf{R}^0_{n-1,n}\mathbf{x} = d\mathbf{e}_1$. Here the kth component vanishes by the rotation $R_{k-1,k}$.

Finally, there are several algorithms for computing the Givens rotations that improve over the straightforward evaluation of $\mathbf{R}^0_{ij}\mathbf{x}$. A robust algorithm minimizing the loss of precision is given in Björck (1996). An algorithm minimizing memory requirements was proposed by Stewart (1976). On the other hand, Gentleman (1973) and Hammarling (1974) proposed modifications aiming to minimize the number of arithmetic operations.

QR Decomposition by Householder Reflections or Givens Rotations

An appropriate combination of HRs or GRs, respectively, can be used to compute the QR decomposition of a given matrix $\mathbf{A} \in \mathbb{R}^{m\times n}$, $m \geq n$, in a following way. Let $\mathbf{Q_i}$, $i = 1, \ldots, n-1$, denote an orthonormal matrix in $\mathbb{R}^{m\times m}$ such that premultiplication of $\mathbf{B} = \mathbf{Q_{i-1}} \cdots \mathbf{Q_1 A}$ by $\mathbf{Q_i}$ can zero all elements in the ith column that are below the diagonal and such that the previous columns $1, \ldots, i-1$ are not affected at all. Such a matrix can be a blockwise diagonal matrix with blocks being the identity matrix $\mathbf{I_{i-1}}$ and a matrix $\mathbf{M}$ solving Problem 1 for the vector composed of elements in the ith column of $\mathbf{B}$ that lie on and below the diagonal. The first part $\mathbf{I_{i-1}}$ guarantees that the columns $1, \ldots, i-1$ of matrix $\mathbf{B}$ are not affected by multiplication, whereas the second block $\mathbf{M}$ transforms all elements in the ith column that are below the diagonal to zero. Naturally, matrix $\mathbf{M}$ can be found by means of HRs or GRs as described in previous paragraphs.

This way, we construct a series of matrices $\mathbf{Q_1}, \ldots, \mathbf{Q_n}$ such that

$$\mathbf{Q_n} \cdots \mathbf{Q_1 A} = \begin{pmatrix} \mathbf{R} \\ \mathbf{0} \end{pmatrix}.$$

Since all matrices $\mathbf{Q_1}, \ldots, \mathbf{Q_n}$ are orthonormal, $\mathbf{Q_t} = \mathbf{Q_n} \cdots \mathbf{Q_1}$ is also orthonormal and its inverse equals its transpose: $\mathbf{Q_t^{-1}} = \mathbf{Q_t^\top}$. Hence,

$$\mathbf{A} = (\mathbf{Q_n} \cdots \mathbf{Q_1})^\top \begin{pmatrix} \mathbf{R} \\ \mathbf{0} \end{pmatrix} = \mathbf{Q} \begin{pmatrix} \mathbf{R} \\ \mathbf{0} \end{pmatrix}$$

as described in Theorem 3.

We describe now the QR algorithm using HRs or GRs. Let $\mathbf{M}(\mathbf{x})$ denote the orthonormal matrix from Problem 1 constructed for a vector $\mathbf{x}$ by one of the discussed methods.

Algorithm 3

$\mathbf{Q} = \mathbf{I_m}$
$\mathbf{R} = \mathbf{A}$
`for i = 1 to n`
  $\mathbf{x} = \{R_{ki}\}_{k=i}^{m}$
  $\mathbf{Q_i} = \begin{pmatrix} \mathbf{I_{i-1}} & \mathbf{0} \\ \mathbf{0} & \mathbf{M}(\mathbf{x}) \end{pmatrix}$
  $\mathbf{Q} = \mathbf{Q_i Q}$
  $\mathbf{R} = \mathbf{Q_i R}$
`end`
$\mathbf{Q} = \mathbf{Q}^\top$
$\mathbf{R} = \{R_{ij}\}_{i=1,j=1}^{n,n}$

There are also modifications of this basic algorithm employing pivoting for better numerical performance and even revealing rank of the system, see

Hong and Tan (1992) and Higham (2000) for instance. An error analysis of the QR decomposition by HRs and GRs are given by Gentleman (1975) and Higham (2000), respectively.

Gram-Schmidt Orthogonalization

Given a nonsingular matrix $\mathbf{A} \in \mathbb{R}^{m \times n}, m \geq n$, the Gram-Schmidt orthogonalization constructs a matrix $\mathbf{Q}$ such that the columns of $\mathbf{Q}$ are orthonormal to each other and span the same space as the columns of $\mathbf{A}$. Thus, $\mathbf{A}$ can be expressed as $\mathbf{Q}$ multiplied by another matrix $\mathbf{R}$, whereby the Gram-Schmidt orthogonalization process (GS) ensures that $\mathbf{R}$ is an upper triangular matrix. Consequently, GS can be used to construct the QR decomposition of a matrix $\mathbf{A}$. A survey of GS variants and their properties is given by Björck (1994).

The classical Gram-Schmidt (CGS) process constructs the orthonormal basis stepwise. The first column $\mathbf{Q}_1$ of $\mathbf{Q}$ is simply normalized $\mathbf{A}_1$. Having constructed a orthonormal base $\mathbf{Q}_{1:k} = \{\mathbf{Q}_1, \ldots, \mathbf{Q}_k\}$, the next column $\mathbf{Q}_{k+1}$ is proportional to $\mathbf{A}_{k+1}$ minus its projection to the space span$\{\mathbf{Q}_{1:k}\}$. Thus, $\mathbf{Q}_{k+1}$ is by its definition orthogonal to span$\{\mathbf{Q}_{1:k}\}$, and at the same time, the first k columns of $\mathbf{A}$ and $\mathbf{Q}$ span the same linear space. The elements of the triangular matrix $\mathbf{R}$ from Theorem 3 are then coordinates of the columns of $\mathbf{A}$ given the columns of $\mathbf{Q}$ as a basis.

Algorithm 4

```
for i = 1 to n
    for j = 1 to i - 1
        R_ji = Q_j^⊤ A_i
    end
    Q_i = A_i − Σ_{j=1}^{i−1} R_ji Q_j
    R_ii = (Q_i^⊤ Q_i)^{1/2}
    Q_i = Q_i / R_ii
end
```

Similarly to many decomposition algorithms, also CGS allows a memory efficient implementation since the computed orthonormal columns of $\mathbf{Q}$ can rewrite the original columns of $\mathbf{A}$. Despite this feature and mathematical correctness, the CGS algorithm does not always behave well numerically because numerical errors can very quickly accumulate. For example, an error made in computing $\mathbf{Q}_1$ affects $\mathbf{Q}_2$, errors in both of these terms (although caused initially just by an error in $\mathbf{Q}_1$) adversely influence $\mathbf{Q}_3$ and so on. Fortunately, there is a modified Gram-Schmidt (MGS) procedure, which prevents such an error accumulation by subtracting linear combinations of $\mathbf{Q}_k$ directly from $\mathbf{A}$ before constructing following orthonormal vectors. (Surprisingly, MGS is historically older than CGS.)

Algorithm 5

```
for i = 1 to n
```

```
    Q_i = A_i
    R_ii = (Q_i^T Q_i)^{1/2}
    Q_i = Q_i / R_ii
    for j = i + 1 to n
        R_ji = Q_i^T A_j
        A_j = A_j − R_ij Q_i
    end
end
```

Apart from this algorithm (the row version of MGS), there are also a column version of MGS by Björck (1994) and MGS modifications employing iterative orthogonalization and pivoting by Dax (2000). Numerical superiority of MGS over CGS was experimentally established already by Rice (1966). This result is also theoretically supported by the GS error analysis in Björck (1994), who uncovered numerical equivalence of the QR decompositions done by MGS and HRs.

1.4 Singular Value Decomposition

The singular value decomposition (SVD) plays an important role in numerical linear algebra and in many statistical techniques as well. Using two orthonormal matrices, SVD can diagonalize any matrix $\mathbf{A}$ and the results of SVD can tell a lot about (numerical) properties of the matrix. (This is closely related to the eigenvalue decomposition: any symmetric square matrix $\mathbf{A}$ can be diagonalized, $\mathbf{A} = \mathbf{V}\mathbf{D}\mathbf{V}^\top$, where $\mathbf{D}$ is a diagonal matrix containing the eigenvalues of $\mathbf{A}$ and $\mathbf{V}$ is an orthonormal matrix.)

Theorem 5. *Let $\mathbf{A} \in \mathbb{R}^{m\times n}$ be a matrix of rank r. Then there exist orthonormal matrices $\mathbf{U} \in \mathbb{R}^{m\times m}$ and $\mathbf{V} \in \mathbb{R}^{n\times n}$ and a diagonal matrix $\mathbf{D} \in \mathbb{R}^{m\times n}$, with the diagonal elements $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_r > \sigma_{r+1} = \ldots = \sigma_{\min\{m,n\}} = 0$, such that $\mathbf{A} = \mathbf{U}\mathbf{D}\mathbf{V}^\top$.*

Numbers $\sigma_1, \ldots, \sigma_{\min\{m,n\}}$ represent the singular values of $\mathbf{A}$. Columns $\mathbf{U}_i$ and $\mathbf{V}_i$ of matrices $\mathbf{U}$ and $\mathbf{V}$ are called the left and right singular vectors of $\mathbf{A}$ associated with singular value σ_i, respectively, because $\mathbf{A}\mathbf{V}_i = \sigma_i\mathbf{U}_i$ and $\mathbf{U}_i^\top\mathbf{A} = \sigma_i\mathbf{V}_i^\top, i = 1, \ldots, \min\{m, n\}$.

Similarly to the QR decomposition, SVD offers a numerically stable way to solve a system of linear equations. Given a system $\mathbf{A}\mathbf{x} = \mathbf{U}\mathbf{D}\mathbf{V}^\top\mathbf{x} = \mathbf{b}$, one can transform it to $\mathbf{U}^\top\mathbf{A}\mathbf{x} = \mathbf{D}\mathbf{V}^\top\mathbf{x} = \mathbf{U}^\top\mathbf{b}$ and solve it in two trivial steps: first, finding a solution $\mathbf{z}$ of $\mathbf{D}\mathbf{z} = \mathbf{U}^\top\mathbf{b}$, and second, setting $\mathbf{x} = \mathbf{V}\mathbf{z}$, which is equivalent to $\mathbf{V}^\top\mathbf{x} = \mathbf{z}$.

On the other hand, the power of SVD lies in its relation to many important matrix properties; see Trefethen and Bau (1997), for instance. First of all, the singular values of a matrix $\mathbf{A}$ are equal to the (positive) square roots of the eigenvalues of $\mathbf{A}^\top\mathbf{A}$ and $\mathbf{A}\mathbf{A}^\top$, whereby the associated left and right singular vectors are identical with the corresponding eigenvectors. Thus, one

can compute the eigenvalues of $\mathbf{A}^\top\mathbf{A}$ directly from the original matrix $\mathbf{A}$. Second, the number of nonzero singular values equals the rank of a matrix. Consequently, SVD can be used to find an effective rank of a matrix, to check a near singularity and to compute the condition number of a matrix. That is, it allows to assess conditioning and sensitivity to errors of a given system of equations. Finally, let us note that there are far more uses of SVD: identification of the null space of $\mathbf{A}$, $\text{null}(\mathbf{A}) = \text{span}\{\mathbf{V}_{k+1}, \ldots, \mathbf{V}_n\}$; computation of the matrix pseudo-inverse, $\mathbf{A}^- = \mathbf{V}\mathbf{D}^-\mathbf{U}^\top$; low-rank approximations and so on. See Björck (1996) and Trefethen and Bau (1997) for details.

Let us now present an overview of algorithms for computing the SVD decomposition, which are not described in details due to their extent. The first stable algorithm for computing the SVD was suggested by Golub and Kahan (1965). It involved reduction of a matrix $\mathbf{A}$ to its bidiagonal form by HRs, with singular values and vectors being computed as eigenvalues and eigenvectors of a specific tridiagonal matrix using a method based on Sturm sequences. The final form of the QR algorithm for computing SVD, which has been the preferred SVD method for dense matrices up to now, is due to Golub and Reinsch (1970); see Anderson et al. (1999), Björck (1996) or Gentle (1998) for the description of the algorithm and some modifications. An alternative approach based on Jacobi algorithm was given by Hari and Veselić (1987). Latest contributions to the pool of computational methods for SVD, including von Matt (1995), Demmel et al. (1999) and Higham (2000), aim to improve the accuracy of singular values and computational speed using recent advances in the QR decomposition.

1.5 Matrix Inversion

In previous sections, we described how matrix decompositions can be used for solving systems of linear equations. Let us now discuss the use of matrix decompositions for inverting a nonsingular squared matrix $\mathbf{A} \in \mathbb{R}^{n\times n}$, although matrix inversion is not needed very often. All discussed matrix decomposition construct two or more matrices $\mathbf{A_1}, \ldots, \mathbf{A_d}$ such that $\mathbf{A} = \mathbf{A_1} \cdot \ldots \cdot \mathbf{A_d}$, where matrices $\mathbf{A_l}, l = 1, \ldots, d$, are orthonormal, triangular, or diagonal. Because $\mathbf{A}^{-1} = \mathbf{A_d}^{-1} \cdot \ldots \cdot \mathbf{A_1}^{-1}$, we just need to be able to invert orthonormal and triangular matrices (a diagonal matrix is a special case of a triangular matrix).

First, an orthonormal matrix $\mathbf{Q}$ satisfies by definition $\mathbf{Q}^\top\mathbf{Q} = \mathbf{Q}\mathbf{Q}^\top = \mathbf{I_n}$. Thus, inversion is in this case equivalent to the transposition of a matrix: $\mathbf{Q}^{-1} = \mathbf{Q}^\top$.

Second, inverting an upper triangular matrix $\mathbf{U}$ can be done by solving directly $\mathbf{X}\mathbf{U} = \mathbf{I_n}$, which leads to the backward substitution method. Let $\mathbf{X} = \{X_{ij}\}_{i=1,j=1}^{n,n}$ denote the searched for inverse matrix $\mathbf{U}^{-1}$.

Algorithm 6
$\mathbf{X} = \mathbf{0_n}$

```
for i = n to 1
    X_ii = 1/U_ii
    for j = i + 1 to n
        X_ij = -( sum_{k=i+1}^{j} X_kj U_ik ) / U_jj
    end
end
```

The inversion of a lower triangular matrix $\mathbf{L}$ can be done analogously: the algorithm is applied to $\mathbf{L}^\top$, that is, U_{ij} is replaced by L_{ji} for $i, j = 1, \ldots, n$.

There are several other algorithms available such as forward substitution or blockwise inversion. Designed for a faster and more (time) efficient computation, their numerical behavior does not significantly differ from the presented algorithm. See Croz and Higham (1992) for an overview and numerical study.

2 Direct Methods for Solving Linear Systems

A system of linear equations can be written in the matrix notation as

$$\mathbf{A}\mathbf{x} = \mathbf{b}, \tag{6}$$

where $\mathbf{A}$ denotes the coefficient matrix, $\mathbf{b}$ is the right-hand side, and $\mathbf{x}$ represents the solution vector we search for. The system (6) has a solution if and only if $\mathbf{b}$ belongs to the vector space spanned by the columns of $\mathbf{A}$.

- If $m < n$, that is, the number of equations is smaller than the number of unknown variables, or if $m \geq n$ but $\mathbf{A}$ does not have a full rank (which means that some equations are linear combinations of the other ones), the system is underdetermined and there are either no solution at all or infinitely many of them. In the latter case, any solution can be written as a sum of a particular solution and a vector from the nullspace of $\mathbf{A}$. Finding the solution space can involve the SVD decomposition (Section 1.4).
- If $m > n$ and the matrix $\mathbf{A}$ has a full rank, that is, if the number of equations is greater than the number of unknown variables, there is generally no solution and the system is overdetermined. One can search some $\mathbf{x}$ such that the distance between $\mathbf{A}\mathbf{x}$ and $\mathbf{b}$ is minimized, which leads to the linear least-squares problem if distance is measured by L_2 norm; see Chapter **??**.
- If $m = n$ and the matrix $\mathbf{A}$ is nonsingular, the system (6) has a unique solution. Methods suitable for this case will be discussed in the rest of this section as well as in Section 3.

¿From here on, we concentrate on systems of equations with unique solutions.

There are two basic classes of methods for solving system (6). The first class is represented by direct methods. They theoretically give an exact solution in a (predictable) finite number of steps. Unfortunately, this does not have

to be true in computational praxis due to rounding errors: an error made in one step spreads in all following steps. Classical direct methods are discussed in this section. Moreover, solving an equation system by means of matrix decompositions, as discussed in Section 1, can be classified as a direct method as well. The second class is called iterative methods, which construct a series of solution approximations that (under some assumptions) converges to the solution of the system. Iterative methods are discussed in Section 3. Finally, note that some methods are on the borderline between the two classes; for example, gradient methods (Section 3.5) and iterative refinement (Section 2.2).

Further, the direct methods discussed in this section are not necessarily optimal for an arbitrary system (6). Let us deal with the main exceptions. First, even if a unique solution exist, numerical methods can fail to find the solution: if the number of unknown variables n is large, rounding errors can accumulate and result in a wrong solution. The same applies very much to systems with a nearly singular coefficient matrix. One alternative is to use iterative methods (Section 3), which are less sensitive to these problems. Another approach is to use the QR or SVD decompositions (Section 1), which can transform some nearly singular problems to nonsingular ones. Second, very large problems including hundreds or thousands of equations and unknown variables may be very time demanding to solve by standard direct methods. On the other hand, their coefficient matrices are often sparse, that is, most of their elements are zeros. Special strategies to store and solve such problems are discussed in Section 5.

To conclude these remarks, let us mention a close relation between solving the system (6) and computing the inverse matrix $\mathbf{A}^{-1}$:

- having an algorithm that for a matrix $\mathbf{A}$ computes $\mathbf{A}^{-1}$, we can find the solution to (6) as $\mathbf{x} = \mathbf{A}^{-1}\mathbf{b}$;
- an algorithm solving the system (6) can be used to compute $\mathbf{A}^{-1}$ as follows. Solve n linear systems $\mathbf{A}\mathbf{x}_i = \mathbf{e}_i$, $i = 1, \ldots, n$ (or the corresponding system with multiple right-hand sides), where $\mathbf{e}_i$ denotes the ith unit vector. Then $\mathbf{A}^{-1} = (\mathbf{x}_1, \ldots, \mathbf{x}_n)$.

In the rest of this section, we concentrate on the Gauss-Jordan elimination (Section 2.1) and its modifications and extensions, such as iterative refinement (Section 2.2). A wealth of information on direct methods can be found in monographs Axelsson (1994), Gentle (1998) and Golub and van Loan (1996).

2.1 Gauss-Jordan Elimination

In this subsection, we will simultaneously solve the linear systems

$$\mathbf{A}\mathbf{x}_1 = \mathbf{b}_1, \quad \mathbf{A}\mathbf{x}_2 = \mathbf{b}_2, \quad \ldots, \quad \mathbf{A}\mathbf{x}_k = \mathbf{b}_k$$

and a matrix equation $\mathbf{A}\mathbf{X} = \mathbf{B}$, where $\mathbf{X}, \mathbf{B} \in \mathbb{R}^{n \times l}$ (its solution is $\mathbf{X} = \mathbf{A}^{-1}\mathbf{B}$, yielding the inverse $\mathbf{A}^{-1}$ for a special choice $\mathbf{B} = \mathbf{I_n}$). They can be written as a linear matrix equation

$$\mathbf{A}[\mathbf{x}_1|\mathbf{x}_2|\ldots|\mathbf{x}_k|\mathbf{X}] = [\mathbf{b}_1|\mathbf{b}_2|\ldots|\mathbf{b}_k|\mathbf{B}], \tag{7}$$

where the operator | stands for column augmentation.

The Gauss-Jordan elimination (GJ) is based on elementary operations that do not affect the solution of an equation system. The solution of (7) will not change if we perform any of the following operations:

- interchanging any two rows of $\mathbf{A}$ and the corresponding rows of $\mathbf{b}_i$'s and $\mathbf{B}$, $i = 1, \ldots, k$;
- multiplying a row of $\mathbf{A}$ and the same row of $\mathbf{b}_i$'s and $\mathbf{B}$ by a nonzero number, $i = 1, \ldots, k$;
- adding to a chosen row of $\mathbf{A}$ and the same row of $\mathbf{b}_i$'s and $\mathbf{B}$ a linear combination of other rows, $i = 1, \ldots, k$.

Interchanging any two columns of $\mathbf{A}$ is possible too, but it has to be followed by interchanging the corresponding rows of all solutions $\mathbf{x}_i$ and $\mathbf{X}$ as well as of right sides $\mathbf{b}_i$ and $\mathbf{B}$, $i = 1, \ldots, k$. Each row or column operation described above is equivalent to the pre- or postmultiplication of the system by a certain elementary matrix $\mathbf{R}$ or $\mathbf{C}$, respectively, that are results of the same operation applied to the identity matrix $\mathbf{I_n}$.

GJ is a technique that applies one or more of these elementary operations to (7) so that $\mathbf{A}$ becomes the identity matrix $\mathbf{I_n}$. Simultaneously, the right-hand side becomes the set of solutions. Denoting $\mathbf{R_i}, i = 1, \ldots, O$, the matrices corresponding to the ith row operation, the combination of all operations has to constitute inverse $\mathbf{A}^{-1} = \mathbf{R_O} \cdot \ldots \cdot \mathbf{R_3}\mathbf{R_2}\mathbf{R_1}$ and hence $\mathbf{x} = \mathbf{R_O} \cdot \ldots \cdot \mathbf{R_3}\mathbf{R_2}\mathbf{R_1}\mathbf{b}$. The exact choice of these elementary operation is described in the following paragraph.

Pivoting in Gauss-Jordan Elimination

Let us now discuss several well-known variants of the Gauss-Jordan elimination. GJ without pivoting does not interchange any rows or columns; only multiplication and addition of rows are permitted. First, nonzero nondiagonal elements in the first column $\mathbf{A}_1$ are eliminated: the first row of (7) is divided by its diagonal element A_{11} and the A_{i1}-multiple of the modified first row is subtracted from the ith row, $i = 2, \ldots, n$. We can proceed the same way for all n columns of $\mathbf{A}$, and thus, transform $\mathbf{A}$ to the identity matrix $\mathbf{I_n}$. It is easy to see that the method fails if the diagonal element in a column to be eliminated, the so-called pivot, is zero in some step. Even if this is not the case, one should be aware that GJ without pivoting is numerically unstable.

On the other hand, the GJ method becomes stable when using pivoting. This means that one can interchange rows (partial pivoting) or rows and columns (full pivoting) to put a suitable matrix element to the position of the current pivot. Since it is desirable to keep the already constructed part of the identify matrix, only rows below and columns right to the current pivot are considered. GJ with full pivoting is numerically stable. From the application

point of view, GJ with partial pivoting is numerically stable too, although there are artificial examples where it fails. Additionally, the advantage of partial pivoting (compared to full pivoting) is that it does not change the order of solution components.

There are various strategies to choose a pivot. A very good choice is the largest available element (in absolute value). This procedure depends however on the original scaling of the equations. Implicit pivoting takes scaling into account and chooses a pivot as if the original system were rescaled so that the largest element of each row would be equal to one.

Finally, let us add several concluding remarks on efficiency of GJ and its relationship to matrix decompositions. As shown, GJ can efficiently solve problems with multiple right-hand sides known in advance and compute $\mathbf{A}^{-1}$ at the same time. On the other hand, if it is necessary to solve later a new system with the same coefficient matrix $\mathbf{A}$ but a new right-hand side $\mathbf{b}$, one has to start the whole elimination process again, which is time demanding, or compute $\mathbf{A}^{-1}\mathbf{b}$ using the previously computed inverse matrix $\mathbf{A}^{-1}$, which leads to further error accumulation. In praxis, one should prefer matrix decompositions, which do not have this drawback. Specifically, the LU decomposition (Section 1.2) is equivalent to GJ (with the same kind of pivoting applied in both cases) and allows us to repeatedly solve systems with the same coefficient matrix in an efficient way.

2.2 Iterative Refinement

In the introduction to Section 2, we noted that direct methods are rather sensitive to rounding errors. Iterative refinement offers a way to improve the solution obtained by any direct method, unless the system matrix $\mathbf{A}$ is too ill-conditioned or even singular.

Let $\mathbf{x}_1$ denote an initially computed (approximate) solution of (6). Iterative refinement is a process constructing a series $\mathbf{x}_i$, $i = 1, 2, \ldots$, as described in Algorithm 7. First, given a solution $\mathbf{x}_i$, the residuum $\mathbf{r}_i = \mathbf{A}\mathbf{x}_i - \mathbf{b}$ is computed. Then, one obtains the correction $\boldsymbol{\Delta}\mathbf{x}_i$ by solving the original system with residuum $\mathbf{r}_i$ on the right-hand side.

Algorithm 7

Repeat for $i = 1, 2, \ldots$
  compute $\mathbf{r}_i = \mathbf{b} - \mathbf{A}\mathbf{x}_i$
  solve $\mathbf{A}\boldsymbol{\Delta}\mathbf{x}_i = \mathbf{r}_i$ for $\boldsymbol{\Delta}\mathbf{x}_i$
  set $\mathbf{x}_{i+1} = \mathbf{x}_i + \boldsymbol{\Delta}\mathbf{x}_i$
until the desired precision is achieved.

It is reasonable to carry out the computation of residuals $\mathbf{r}_i$ in a higher precision because a lot of cancellation occurs if $\mathbf{x}_i$ is a good approximation. Nevertheless, provided that the coefficient matrix $\mathbf{A}$ is not too ill-conditioned, Skeel (1980) proved that GJ with partial pivoting and only one step of iterative refinement computed in a fixed precision is stable (it has a relative backward

error proportional to the used precision). In spite of this result, one can recommend to use iterative refinement repeatedly until the desired precision is reached.

Additionally, an important feature of iterative refinement is its low computational costs. Provided that a system is solved by means of decompositions (e.g., GJ is implemented as the LU decomposition), a factorization of $\mathbf{A}$ is available already after computing the initial solution $\mathbf{x}_1$. Subsequently, solving any system with the same coefficient matrix $\mathbf{A}$, such as $\mathbf{A}\boldsymbol{\Delta}\mathbf{x}_i = \mathbf{r}_i$, can be done fast and efficiently and the computational costs of iterative refinement are small.

3 Iterative Methods for Solving Linear Systems

Direct methods for solving linear systems theoretically give the exact solution in a finite number of steps, see Section 2. Unfortunately, this is rarely true in applications because of rounding errors: an error made in one step spreads further in all following steps! Contrary to direct methods, iterative methods construct a series of solution approximations such that it converges to the exact solution of a system. Their main advantage is that they are self-correcting, see Section 3.1.

In this section, we first discuss general principles of iterative methods that solve linear system (6), $\mathbf{A}\mathbf{x} = \mathbf{b}$, whereby we assume that $\mathbf{A} \in \mathbb{R}^{n\times n}$ and the system has exactly one solution $\mathbf{x_e}$ (see Section 2 for more details on other cases). Later, we describe most common iterative methods: the Jacobi, Gauss-Seidel, successive overrelaxation, and gradient methods (Sections 3.2–3.5). Monographs containing detailed discussion of these methods include Björck (1996), Golub and van Loan (1996) and Hackbusch (1994). Although we treat these methods separately from the direct methods, let us mention here that iterative methods can usually benefit from a combination with the Gauss elimination, see Milaszewicz (1987) and Allanelli and Hadjidimos (2004), for instance.

To unify the presentation of all methods, let $\mathbf{D}$, $\mathbf{L}$, and $\mathbf{U}$ denote the diagonal, lower triangular and upper triangular parts of a matrix $\mathbf{A}$ throughout this section:

$$D_{ij} = \begin{cases} A_{ij} & \text{for } i = j, \\ 0 & \text{otherwise;} \end{cases} \qquad L_{ij} = \begin{cases} A_{ij} & \text{for } i > j, \\ 0 & \text{otherwise;} \end{cases} \qquad U_{ij} = \begin{cases} A_{ij} & \text{for } i < j, \\ 0 & \text{otherwise.} \end{cases}$$

3.1 General Principle of Iterative Methods for Linear Systems

An iterative method for solving a linear system $\mathbf{A}\mathbf{x} = \mathbf{b}$ constructs an iteration series $\mathbf{x}_i$, $i = 0, 1, 2, \ldots$, that under some conditions converges to the exact solution $\mathbf{x_e}$ of the system ($\mathbf{A}\mathbf{x_e} = \mathbf{b}$). Thus, it is necessary to choose a starting point $\mathbf{x}_0$ and iteratively apply a rule that computes $\mathbf{x}_{i+1}$ from an already known $\mathbf{x}_i$.

A starting vector $\mathbf{x}_0$ is usually chosen as some approximation of $\mathbf{x}$. (Luckily, its choice cannot cause divergence of a convergent method.) Next, given $\mathbf{x}_i, i \in \mathbb{N}$, the subsequent element of the series is computed using a rule of the form

$$\mathbf{x}_{i+1} = \mathbf{B_i}\mathbf{x}_i + \mathbf{C_i}\mathbf{b}, \quad i = 0, 1, 2, \ldots, \tag{8}$$

where $\mathbf{B_i}, \mathbf{C_i} \in \mathbb{R}^{n \times n}, i \in \mathbb{N}$, are matrix series. Different choices of $\mathbf{B}_i$ and $\mathbf{C_i}$ define different iterative methods.

Let us discuss now a minimal set of conditions on $\mathbf{B_i}$ and $\mathbf{C_i}$ in (8) that guarantee the convergence of an iterative method. First of all, it has to hold that $\mathbf{B_i} + \mathbf{C_i}\mathbf{A} = \mathbf{I_n}$ for all $i \in \mathbb{N}$, or equivalently,

$$\mathbf{x_e} = \mathbf{B_i}\mathbf{x_e} + \mathbf{C_i}\mathbf{b} = (\mathbf{B_i} + \mathbf{C_i}\mathbf{A})\mathbf{x_e}, \qquad i \in \mathbb{N}. \tag{9}$$

In other words, once the iterative process reaches the exact solution $\mathbf{x_e}$, all consecutive iterations should stay equal to $\mathbf{x_e}$ and the method cannot depart from this solution. Second, starting from a point $\mathbf{x}_0 \neq \mathbf{x_e}$, we have to ensure that approximations $\mathbf{x}_i$ will converge to $\mathbf{x_e}$ as i increases.

Theorem 6. *An iteration series $\mathbf{x}_i$ given by (8) converges to the solution of system (6) for any chosen $\mathbf{x}_0$ iff*

$$\lim_{i \to \infty} \mathbf{B_i}\mathbf{B_{i-1}} \ldots \mathbf{B_0} = \mathbf{0}.$$

In praxis, stationary iterative methods are used, that is, methods with constant $\mathbf{B_i} = \mathbf{B}$ and $\mathbf{C_i} = \mathbf{C}$, $i \in \mathbb{N}$. Consequently, an iteration series is then constructed using

$$\mathbf{x}_{i+1} = \mathbf{B}\mathbf{x}_i + \mathbf{C}\mathbf{b}, \quad i = 0, 1, 2, \ldots \tag{10}$$

and the convergence condition in Theorem 6 has a simpler form.

Theorem 7. *An iteration series $\mathbf{x}_i$ given by (10) converges to the solution of system (6) for any chosen $\mathbf{x}_0$ iff the spectral radius $\rho(\mathbf{B}) < 1$, where $\rho(\mathbf{B}) = \max_{i=1,\ldots,n} |\lambda_i|$ and $\lambda_1, \ldots, \lambda_n$ represent the eigenvalues of $\mathbf{B}$.*

Note that the convergence condition $\rho(\mathbf{B}) < 1$ holds, for example, if $\|\mathbf{B}\| < 1$ in any matrix norm. Moreover, Theorem 7 guarantees the self-correcting property of iterative methods since convergence takes place independent of the starting value $\mathbf{x}_0$. Thus, if computational errors adversely affect $\mathbf{x}_i$ during the ith iteration, $\mathbf{x}_i$ can be considered as a new starting vector and the iterative method will further converge. Consequently, the iterative methods are in general more robust than the direct ones.

Apparently, such an iterative process can continue arbitrarily long unless $\mathbf{x}_i = \mathbf{x_e}$ at some point. This is impractical and usually unnecessary. Therefore, one uses stopping (or convergence) criteria that stop the iterative process when a pre-specified condition is met. Commonly used stopping criteria are

based on the change of the solution or residual vector achieved during one iteration. Specifically, given a small $\varepsilon > 0$, the iterative process is stopped after the ith iteration when $\|\mathbf{x}_i - \mathbf{x}_{i-1}\| \leq \varepsilon$, $\|\mathbf{r}_i - \mathbf{r}_{i-1}\| \leq \varepsilon$, or $\|\mathbf{r}_i\| \leq \varepsilon$, where $\mathbf{r}_i = \mathbf{A}\mathbf{x}_i - \mathbf{b}$ is a residual vector. Additionally, a maximum acceptable number of iterations is usually specified.

3.2 Jacobi Method

The Jacobi method is motivated by the following observation. Let $\mathbf{A}$ have nonzero diagonal elements (the rows of any nonsingular matrix can be reorganized to achieve this). Then the diagonal part $\mathbf{D}$ of $\mathbf{A}$ is nonsingular and the system (6) can be rewritten as $\mathbf{D}\mathbf{x} + (\mathbf{L} + \mathbf{U})\mathbf{x} = \mathbf{b}$. Consequently,

$$\mathbf{x} = \mathbf{D}^{-1}[(-\mathbf{L} - \mathbf{U})\mathbf{x} + \mathbf{b}].$$

Replacing $\mathbf{x}$ on the left-hand side by $\mathbf{x}_{i+1}$ and $\mathbf{x}$ on the right-hand side by $\mathbf{x}_i$ leads to the iteration formula of the Jacobi method:

$$\mathbf{x}_{i+1} = -\mathbf{D}^{-1}(\mathbf{L} + \mathbf{U})\mathbf{x}_i + \mathbf{D}^{-1}\mathbf{b}.$$

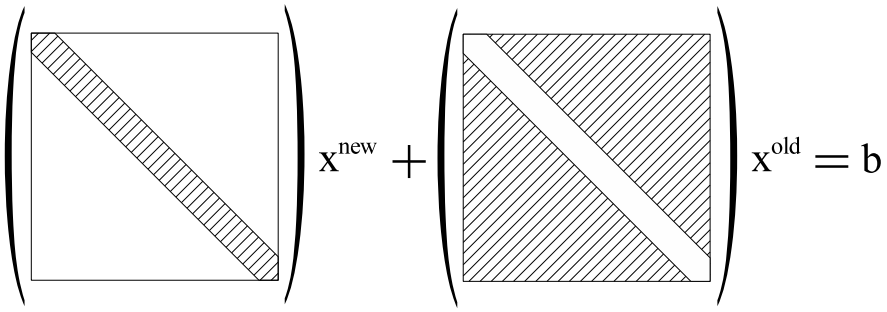

Fig. 4. Scheme of the Jacobi method

The intuition of the Jacobi method is very simple: given an approximation $\mathbf{x}^{old}$ of the solution, let us express the kth component x_k of $\mathbf{x}$ as a function of the other components from the kth equation and compute x_k given $\mathbf{x}^{old}$:

$$x_k^{new} = \frac{1}{A_{kk}} \left(b_k - \sum_{\substack{j=1 \\ j \neq k}}^{n} A_{kj} x_j^{old} \right), \tag{11}$$

$k = 1, \ldots, n$ (see Figure 4).

The Jacobi method converges for any starting vector $\mathbf{x}_0$ as long as $\rho(\mathbf{D}^{-1}(\mathbf{L}+\mathbf{U})) < 1$, see Theorem 7. This condition is satisfied for a relatively big class of matrices including diagonally dominant matrices (matrices $\mathbf{A}$ such that $\sum_{j=1,j\neq i}^{n} |A_{ij}| \leq |A_{ii}|$ for $i = 1, \ldots, n$), and symmetric matrices $\mathbf{A}$ such that $\mathbf{D}$, $\mathbf{A} = \mathbf{L}+\mathbf{D}+\mathbf{U}$, and $-\mathbf{L}+\mathbf{D}-\mathbf{U}$ are all positive definite. Although there are many improvements to the basic principle of the Jacobi method in terms of convergence to $\mathbf{x_e}$, see Sections 3.3 and 3.4, its advantage is an easy and fast implementation (elements of a new iteration $\mathbf{x}_i$ can be computed independently of each other).

3.3 Gauss-Seidel Method

Analogously to the Jacobi method, we can rewrite system (6) as $(\mathbf{L}+\mathbf{D})\mathbf{x} + \mathbf{U}\mathbf{x} = \mathbf{b}$, which further implies $\mathbf{x} = (\mathbf{L}+\mathbf{D})^{-1}[-\mathbf{U}\mathbf{x}+\mathbf{b}]$. This leads to the iteration formula of the Gauss-Seidel method:

$$\mathbf{x}_{i+1} = -(\mathbf{L}+\mathbf{D})^{-1}\mathbf{U}\mathbf{x}_i + (\mathbf{L}+\mathbf{D})^{-1}\mathbf{b}. \tag{12}$$

The main difference to the Jacobi methods lies in a more efficient use of (11). When computing the kth element x_k^{new}, the first $k-1$ elements $x_1^{new}, \ldots, x_{k-1}^{new}$ are already known (and presumably more precise than $x_1^{old}, \ldots, x_{k-1}^{old}$). Thus, it is possible to use these new values instead of the old ones and speed up the convergence (see Figure 5 for a scheme). Moreover, using this strategy, the newly computed elements of $\mathbf{x}_{i+1}$ can directly overwrite the respective elements of $\mathbf{x}_i$ and save memory this way.

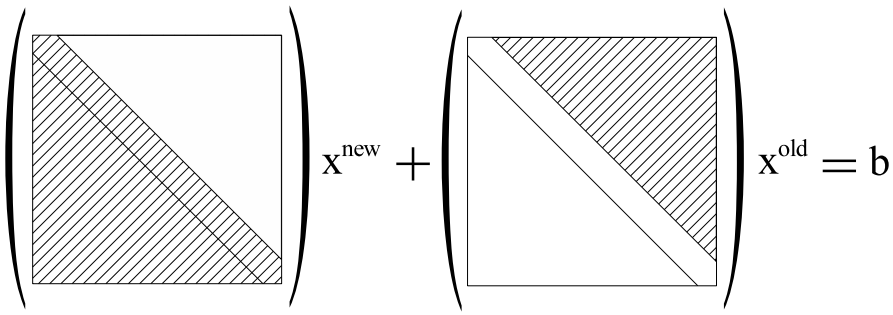

Fig. 5. Scheme of the Gauss-Seidel method

Following the Theorem 7, the Gauss-Seidel method converges for any starting vector $\mathbf{x}_0$ iff $\rho((\mathbf{L}+\mathbf{D})^{-1}\mathbf{U}) < 1$. This condition holds, for example, for diagonally dominant matrices as well as for positive definite ones.

3.4 Successive Overrelaxation Method

The successive overrelaxation (SOR) method aims to further refine the Gauss-Seidel method. The Gauss-Seidel formula (12) can be rewritten as

$$\mathbf{x}_{i+1} = \mathbf{x}_i - \mathbf{D}^{-1}[\{\mathbf{L}\mathbf{x}_{i+1} + (\mathbf{D}+\mathbf{U})\mathbf{x}_i\} - \mathbf{b}] = \mathbf{x}_i - \boldsymbol{\Delta}_\mathbf{i},$$

which describes the difference $\boldsymbol{\Delta}_\mathbf{i}$ between $\mathbf{x}_{i+1}$ and $\mathbf{x}_i$ expressed for the kth element of $\mathbf{x}_{i(+1)}$ from the kth equation, $k = 1, \ldots, n$. The question SOR poses is whether the method can converge faster if we "overly" correct $\mathbf{x}_{i+1}$ in each step; that is, if $\mathbf{x}_i$ is corrected by a multiple ω of $\boldsymbol{\Delta}_\mathbf{i}$ in each iteration. This idea leads to the SOR formula:

$$\mathbf{x}_{i+1} = \mathbf{x}_i - \omega\mathbf{D}^{-1}[\{\mathbf{L}\mathbf{x}_{i+1} + (\mathbf{D}+\mathbf{U})\mathbf{x}_i\} - \mathbf{b}],$$

or in the form (10),

$$\mathbf{x}_{i+1} = (\mathbf{D}+\omega\mathbf{L})^{-1}\{(1-\omega)\mathbf{D} - \omega\mathbf{U}\}\mathbf{x}_i + \omega(\mathbf{D}+\omega\mathbf{L})^{-1}\mathbf{b}. \qquad (13)$$

The parameter ω is called the (over)relaxation parameter and it can be shown that SOR converges only for $\omega \in (0, 2)$, a result derived by Kahan (1958).

A good choice of parameter ω can speed up convergence, as measured by the spectral radius of the corresponding iteration matrix $\mathbf{B}$ (see Theorem 7; a lower spectral radius $\rho(\mathbf{B})$ means faster convergence). There is a choice of literature devoted to the optimal setting of relaxation parameter: see Hadjidimos (2000) for a recent overview of the main results concerning SOR. We just present one important result, which is due to Young (1954).

Definition 1. *A matrix* $\mathbf{A}$ *is said to be two-cyclic consistently ordered if the eigenvalues of the matrix* $\mathbf{M}(\alpha) = \alpha\mathbf{D}^{-1}\mathbf{L} + \alpha^{-1}\mathbf{D}^{-1}\mathbf{U}$*,* $\alpha \neq 0$*, are independent of* α*.*

Theorem 8. *Let the matrix* $\mathbf{A}$ *be two-cyclic consistently ordered. Let the respective Gauss-Seidel iteration matrix* $\mathbf{B} = -(\mathbf{L}+\mathbf{D})^{-1}\mathbf{U}$ *have the spectral radius* $\rho(\mathbf{B}) < 1$*. Then the optimal relaxation parameter* ω *in SOR is given by*

$$\omega_{opt} = \frac{2}{1 + \sqrt{1 - \rho(\mathbf{B})}}$$

and for this optimal value it holds $\rho(\mathbf{B}; \omega_{opt}) = \omega_{opt} - 1$*.*

Using SOR with the optimal relaxation parameter significantly increases the rate of convergence. Note however that the convergence acceleration is obtained only for ω very close to ω_{opt}. If ω_{opt} cannot be computed exactly, it is better to take ω slightly larger rather than smaller. Golub and van Loan (1996) describe an approximation algorithm for $\rho(\mathbf{B})$.

On the other hand, if the assumptions of Theorem 8 are not satisfied, one can employ the symmetric SOR (SSOR), which performs the SOR iteration

twice: once as usual, see (13), and once with interchanged $\mathbf{L}$ and $\mathbf{U}$. SSOR requires more computations per iteration and usually converges slower, but it works for any positive definite matrix and can be combined with various acceleration techniques. See Björck (1996) and Hadjidimos (2000) for details.

3.5 Gradient methods

Gradient iterative methods are based on the assumption that $\mathbf{A}$ is a symmetric positive definite matrix $\mathbf{A}$. They use this assumption to reformulate (6) as a minimization problem: $\mathbf{x_e}$ is the only minimum of the quadratic form

$$Q(\mathbf{x}) = \frac{1}{2}\mathbf{x}^\top \mathbf{A}\mathbf{x} - \mathbf{x}^\top \mathbf{b}.$$

Given this minimization problem, gradient methods construct an iteration series of vectors converging to $\mathbf{x_e}$ using the following principle. Having the ith approximation $\mathbf{x}_i$, choose a direction $\mathbf{v}_i$ and find a number α_i such that the new vector

$$\mathbf{x}_{i+1} = \mathbf{x}_i + \alpha_i \mathbf{v}_i$$

is a minimum of $Q(\mathbf{x})$ on the line $\mathbf{x}_i + \alpha \mathbf{v}_i$, $\alpha \in \mathbb{R}$. Various choices of directions $\mathbf{v}_i$ then render different gradient methods, which are in general non-stationary ($\mathbf{v}_i$ changes in each iteration). We discuss here three methods: the Gauss-Seidel (as a gradient method), steepest descent and conjugate gradients methods.

Gauss-Seidel Method as a Gradient Method

Interestingly, the Gauss-Seidel method can be seen as a gradient method for the choice

$$\mathbf{v}_{kn+i} = \mathbf{e}_i, \quad k = 0, 1, 2, \ldots, \quad i = 1, \ldots, n,$$

where $\mathbf{e}_i$ denotes the ith unit vector. The kth Gauss-Seidel iteration corresponds to n subiterations with $\mathbf{v}_{kn+i}$ for $i = 1, \ldots, n$.

Steepest Descent Method

The steepest descent method is based on the direction $\mathbf{v}_i$ given by the gradient of $Q(\mathbf{x})$ at $\mathbf{x}_i$. Denoting the residuum of the ith approximation $\mathbf{r}_i = \mathbf{b} - \mathbf{A}\mathbf{x}_i$, the iteration formula is

$$\mathbf{x}_{i+1} = \mathbf{x}_i + \frac{\mathbf{r}_i^\top \mathbf{r}_i}{\mathbf{r}_i^\top \mathbf{A}\mathbf{r}_i}\mathbf{r}_i,$$

where $\mathbf{r}_i$ represents the direction $\mathbf{v}_i$ and its coefficient is the $Q(\mathbf{x})$-minimizing choice of α_i. By definition, this method reduces $Q(\mathbf{x}_i)$ at each step, but it is not very effective. The conjugate gradient method discussed in the next subsection will usually perform better.

Conjugate Gradient Method

In the conjugate gradient (CG) method proposed by Hestenes and Stiefel (1952), the directions $\mathbf{v}_i$ are generated by the $\mathbf{A}$-orthogonalization of residuum vectors. Given a symmetric positive definite matrix $\mathbf{A}$, $\mathbf{A}$-orthogonalization is a procedure that constructs a series of linearly independent vectors $\mathbf{v}_i$ such that $\mathbf{v}_i^\top \mathbf{A}\mathbf{v}_j = 0$ for $i \neq j$ (conjugacy or $\mathbf{A}$-orthogonality condition). It can be used to solve the system (6) as follows ($\mathbf{r}_i = \mathbf{b} - \mathbf{A}\mathbf{x}_i$ represents residuals).

Algorithm 8

$\mathbf{v}_0 = \mathbf{r}_0 = \mathbf{b} - \mathbf{A}\mathbf{x}_0$
`do`
$\alpha_i = (\mathbf{v}_i^\top \mathbf{r}_i)/(\mathbf{v}_i^\top \mathbf{A}\mathbf{v}_i)$
$\mathbf{x}_{i+1} = \mathbf{x}_i + \alpha_i \mathbf{v}_i$
$\mathbf{r}_{i+1} = \mathbf{r}_i - \alpha_i \mathbf{A}\mathbf{v}_i$
$\beta_i = -(\mathbf{v}_i^\top \mathbf{A}\mathbf{r}_{i+1})/(\mathbf{v}_i^\top \mathbf{A}\mathbf{v}_i)$
$\mathbf{v}_{i+1} = \mathbf{r}_{i+1} + \beta_i \mathbf{v}_i$
`until a stop criterion holds`

An interesting theoretic property of CG is that it reaches the exact solution in at most n steps because there are not more than n ($\mathbf{A}$-)orthogonal vectors. Thus, CG is not a truly iterative method. (This does not have to be the case if $\mathbf{A}$ is a singular or non-square matrix, see Kammerer and Nashed, 1972.) On the other hand, it is usually used as an iterative method, because it can give a solution within the given accuracy much earlier than after n iterations. Moreover, if the approximate solution $\mathbf{x}_n$ after n iterations is not accurate enough (due to computational errors), the algorithm can be restarted with $\mathbf{x}_0$ set to $\mathbf{x}_n$. Finally, let us note that CG is attractive for use with large sparse matrices because it addresses $\mathbf{A}$ only by its multiplication by a vector. This operation can be done very efficiently for a properly stored sparse matrix, see Section 5.

The principle of CG has many extensions that are applicable also for non-symmetric nonsingular matrices: for example, generalized minimal residual, Saad and Schultz (1986); (stabilized) biconjugate gradients, Vorst (1992); or quasi-minimal residual, Freund and Nachtigal (1991).

4 Eigenvalues and Eigenvectors

In this section, we deal with methods for computing eigenvalues and eigenvectors of a matrix $\mathbf{A} \in \mathbb{R}^{n \times n}$. First, we discuss a simple power method for computing one or few eigenvalues (Section 4.1). Next, we concentrate on methods performing the complete eigenanalysis, that is, finding all eigenvalues (the Jacobi, QR, and LR methods in Sections 4.2–4.5). Finally, we briefly describe a way to improve already computed eigenvalues and to find the corresponding eigenvector. Additionally, note that eigenanalysis can be also done by means

of SVD, see Section 1.4. For more details on the described as well as some other methods, one can consult monographs by Gentle (1998), Golub and van Loan (1996), Press et al. (1992) and Stoer and Bulirsch (2002).

Before discussing specific methods, let us describe the principle common to most of them. We assume that $\mathbf{A} \in \mathbb{R}^{n \times n}$ has eigenvalues $|\lambda_1| \geq |\lambda_2| \geq \cdots \geq |\lambda_n|$. To find all eigenvalues, we transform the original matrix $\mathbf{A}$ to a simpler matrix $\mathbf{B}$ such that it is similar to $\mathbf{A}$ (recall that matrices $\mathbf{A}$ and $\mathbf{B}$ are similar if there is a matrix $\mathbf{T}$ such that $\mathbf{B} = \mathbf{T}^{-1}\mathbf{A}\mathbf{T}$). The similarity of $\mathbf{A}$ and $\mathbf{B}$ is crucial since it guarantees that both matrices have the same eigenvalues and their eigenvectors follow simple relation: if $\mathbf{g}$ is an eigenvector of $\mathbf{B}$ corresponding to its eigenvalue λ, then $\mathbf{T}\mathbf{g}$ is an eigenvector of $\mathbf{A}$ corresponding to the same eigenvalue λ.

There are two basic strategies to construct a similarity transformation $\mathbf{B}$ of the original matrix $\mathbf{A}$. First, one can use a series of simple transformations, such as GRs, and eliminate elements of $\mathbf{A}$ one by one (see the Jacobi method, Section 4.2). This approach is often used to transform $\mathbf{A}$ to its tridiagonal or upper Hessenberg forms. (Matrix $\mathbf{B}$ has the upper Hessenberg form if it is an upper triangular except for the first subdiagonal; that is, $A_{ij} = 0$ for $i > j + 1$, where $i, j = 1, \ldots, n$). Second, one can also factorize $\mathbf{A}$ into $\mathbf{A} = \mathbf{F}_L\mathbf{F}_R$ and switch the order of factors, $\mathbf{B} = \mathbf{F}_R\mathbf{F}_L$ (similarity of $\mathbf{A}$ and $\mathbf{B}$ follows from $\mathbf{B} = \mathbf{F}_R\mathbf{F}_L = \mathbf{F}_L^{-1}\mathbf{A}\mathbf{F}_L$). This is used for example by the LR method (Section 4.5). Finally, there are methods combining both approaches.

4.1 Power Method

In its basic form, the power method aims at finding only the largest eigenvalue λ_1 of a matrix $\mathbf{A}$ and the corresponding eigenvector. Let us assume that the matrix $\mathbf{A}$ has a dominant eigenvalue ($|\lambda_1| > |\lambda_2|$) and n linearly independent eigenvectors.

The power method constructs two series c_i and $\mathbf{x}_i, i \in \mathbb{N}$, that converge to λ_1 and to the corresponding eigenvector $\mathbf{g}_1$, respectively. Starting from a vector $\mathbf{x}_0$ that is not orthogonal to $\mathbf{g}_1$, one only has to iteratively compute $\mathbf{A}\mathbf{x}_i$ and split it to its norm c_{i+1} and the normalized vector $\mathbf{x}_{i+1}$, see Algorithm 9. Usually, the Euclidian ($c_{i+1} = \|\mathbf{A}\mathbf{x}_i\|_2$) and maximum ($c_{i+1} = \max_{j=1,\ldots,n} |(\mathbf{A}\mathbf{x}_i)_j|$) norms are used.

Algorithm 9

$i = 0$
`do`
  $i = i + 1$
  $\mathbf{x}_{i+1} = \mathbf{A}\mathbf{x}_i$
  $c_{i+1} = \|\mathbf{A}\mathbf{x}_{i+1}\|$
  $\mathbf{x}_{i+1} = \mathbf{x}_{i+1}/c_{i+1}$
`until a stop criterion holds`

Although assessing the validity of assumptions is far from trivial, one can usually easily recognize whether the method converges from the behaviour of the two constructed series.

Furthermore, the power method can be extended to search also for other eigenvalues; for example, the smallest one and the second largest one. First, if $\mathbf{A}$ is nonsingular, we can apply the power method to $\mathbf{A}^{-1}$ to find the smallest eigenvalue λ_n because $1/\lambda_n$ is the largest eigenvalue of $\mathbf{A}^{-1}$. Second, if we need more eigenvalues and λ_1 is already known, we can use a reduction method to construct a matrix $\mathbf{B}$ that has the same eigenvalues and eigenvectors as $\mathbf{A}$ except for λ_1, which is replaced by zero eigenvalue. To do so, we need to find a (normalized) eigenvector $\mathbf{h}_1$ of $\mathbf{A}^\top$ corresponding to λ_1 ($\mathbf{A}$ and $\mathbf{A}^\top$ have the same eigenvalues) and to set $\mathbf{B} = \mathbf{A} - \lambda_1 \mathbf{h}_1 \mathbf{h}_1^\top$. Naturally, this process can be repeated to find the third and further eigenvalues.

Finally, let us mention that the power method can be used also for some matrices without dominant eigenvalue (e.g., matrices with $\lambda_1 = \cdots = \lambda_p$ for some $1 < p \leq n$). For further extensions of the power method see Sidi (1989), for instance.

4.2 Jacobi Method

For a symmetric matrix $\mathbf{A}$, the Jacobi method constructs a series of orthogonal matrices $\mathbf{R_i}$, $i \in \mathbb{N}$, such that the matrix $\mathbf{T_i} = \mathbf{R_i}^\top \dots \mathbf{R_1}^\top \mathbf{A} \mathbf{R_1} \dots \mathbf{R_i}$ converges to a diagonal matrix $\mathbf{D}$. Each matrix $\mathbf{R_i}$ is a GR matrix defined in (5), whereby the angle α is chosen so that one nonzero element $(\mathbf{T_i})_{jk}$ becomes zero in $\mathbf{T_{i+1}}$. Formulas for computing $\mathbf{R_i}$ given the element (j, k) to be zeroed are described in Gentle (1998), for instance. Once the matrix $\mathbf{A}$ is diagonalized this way, the diagonal of $\mathbf{D}$ contains the eigenvalues of $\mathbf{A}$ and the columns of matrix $\mathbf{R} = \mathbf{R_1} \cdot \ldots \cdot \mathbf{R_i}$ represent the associated eigenvectors.

There are various strategies to choose an element (j, k) which will be zeroed in the next step. The classical Jacobi method chooses the largest off-diagonal element in absolute value and it is known to converge. (Since searching the maximal element is time consuming, various systematic schemes were developed, but their convergence cannot be often guaranteed.) Because the Jacobi method is relatively slow, other methods are usually preferred (e.g., the QR method). On the other hand, it has recently become interesting again because of its accuracy and easy parallelization (Higham, 1997; Zhou and Brent, 2003).

4.3 Givens and Householder Reductions

The Givens and Householder methods use a similar principle as the Jacobi method. A series of GRs or HRs, designed such that they form similarity transformations, is applied to a symmetric matrix $\mathbf{A}$ in order to transformed it to a tridiagonal matrix. (A tridiagonal matrix is the Hessenberg form for symmetric matrices.) This tridiagonal matrix is then subject to one of the iterative methods, such as the QR or LR methods discussed in the following

paragraphs. Formulas for Givens and Householder similarity transformations are given in Press et al. (1992), for instance.

4.4 QR Method

The QR method is one of the most frequently used methods for the complete eigenanalysis of a nonsymmetric matrix, despite the fact that its convergence is not ensured. A typical algorithm proceeds as follows. In the first step, the matrix $\mathbf{A}$ is transformed into a Hessenberg matrix using Givens or Householder similarity transformations (see Sections 1.3 and 4.3). In the second step, this Hessenberg matrix is subject to the iterative process called chasing. In each iteration, similarity transformations, such as GRs, are first used to create nonzero entries in positions $(i+2, i)$, $(i+3, i)$ and $(i+3, i+1)$ for $i = 1$. Next, similarity transformations are repeatedly used to zero elements $(i+2, i)$ and $(i+3, i)$ and to move these "nonzeros" towards the lower right corner of the matrix (i.e., to elements $(i+2, i)$, $(i+3, i)$ and $(i+3, i+1)$ for $i = i+1$). As a result of chasing, one or two eigenvalues can be extracted. If $A_{n,n-1}$ becomes zero (or negligible) after chasing, element $A_{n,n}$ is an eigenvalue. Consequently, we can delete the nth row and column of the matrix and apply chasing to this smaller matrix to find another eigenvalue. Similarly, if $A_{n-1,n-2}$ becomes zero (or negligible), the two eigenvalues of the 2×2 submatrix in the lower right corner are eigenvalues of $\mathbf{A}$. Subsequently, we can delete last two rows and columns and continue with the next iteration.

Since a more detailed description of the whole iterative process goes beyond the extent of this contribution, we refer a reader to Gentle (1998) for a shorter discussion and to Golub and van Loan (1996) and Press et al. (1992) for a more detailed discussion of the QR method.

4.5 LR Method

The LR method is based on a simple observation that decomposing a matrix $\mathbf{A}$ into $\mathbf{A} = \mathbf{F}_L\mathbf{F}_R$ and multiplying the factors in the inverse order results in a matrix $\mathbf{B} = \mathbf{F}_R\mathbf{F}_L$ similar to $\mathbf{A}$. Using the LU decomposing (Section 1.2), the LR method constructs a matrix series $\mathbf{A_i}$ for $i \in \mathbb{N}$, where $\mathbf{A_1} = \mathbf{A}$ and

$$\mathbf{A_i} = \mathbf{L_i}\mathbf{U_i} \quad \Longrightarrow \quad \mathbf{A_{i+1}} = \mathbf{U_i}\mathbf{L_i},$$

where $\mathbf{L_i}$ is a lower triangular matrix and $\mathbf{U_i}$ is an upper triangular matrix with ones on its diagonal. For a wide class of matrices, including symmetric positive definite matrices, $\mathbf{A_i}$ and $\mathbf{L_i}$ are proved to converge to the same lower triangular matrix $\mathbf{L}$, whereby the eigenvalues of $\mathbf{A}$ form the diagonal of $\mathbf{L}$ and are ordered by the decreasing absolute value.

4.6 Inverse Iterations

The method of inverse iterations can be used to improve an approximation λ^* of an eigenvalue λ of a matrix $\mathbf{A}$. The method is based on the fact that the eigenvector $\mathbf{g}$ associated with λ is also an eigenvector of $\tilde{\mathbf{A}} = (\mathbf{A} - \lambda^*\mathbf{I})^{-1}$ associated with the eigenvalue $\tilde{\lambda} = (\lambda - \lambda^*)^{-1}$. For an initial approximation λ^* close to λ, $\tilde{\lambda}$ is the dominant eigenvalue of $\tilde{\mathbf{A}}$. Thus, it can be computed by the power method described in Section 4.1, whereby λ^* could be modified in each iteration in order to improve the approximation of λ.

This method is not very efficient without a good starting approximation, and therefore, it is not suitable for the complete eigenanalysis. On the other hand, the use of the power method makes it suitable for searching of the eigenvector $\mathbf{g}$ associated with λ. Thus, the method of inverse iterations often complements methods for complete eigenanalysis and serves then as a tool for eigenvector analysis. For this purpose, one does not have to perform the iterative improvement of initial λ^*: applying the power method on $\tilde{\mathbf{A}} = (\mathbf{A} - \lambda^*\mathbf{I})^{-1}$ suffices. See Ipsen (1997), Press et al. (1992) and Stoer and Bulirsch (2002) for more details.

5 Sparse Matrices

Numerical problems arising in some applications, such as seemingly unrelated regressions, spatial statistics, or support vector machines (Chapter **??**), are sparse: they often involve large matrices, which have only a small number of nonzero elements. (It is difficult to specify what exactly "small number" is.) From the practical point of view, a matrix is sparse if it has so many zero elements that it is worth to inspect their structure and use appropriate methods to save storage and the number of operations. Some sparse matrices show a regular pattern of nonzero elements (e.g., band matrices), while some exhibit a rather irregular pattern. In both cases, solving the respective problem efficiently means to store and operate on only nonzero elements and to keep the "fill," the number of newly generated nonzero elements, as small as possible.

In this section, we first discuss some of storage schemes for sparse matrices, which could indicate what types of problems can be effectively treated as sparse ones (Section 5.1). Later, we give examples of classical algorithms adopted for sparse matrices (Section 5.2). Monographs introducing a range of methods for sparse matrices include Duff et al. (1989), Hackbusch (1994) and Saad (2003).

5.1 Storage Schemes for Sparse Matrices

To save storage, only nonzero elements of a sparse vector or matrix should be stored. There are various storage schemes, which require approximately from two to five times the number of nonzero elements to store a vector or a matrix.

Unfortunately, there is no standard scheme. We discuss here the widely used and sufficiently general compressed (row) storage for vectors and for general and banded matrices.

The compressed form of a vector $\mathbf{x}$ consists of a triplet $(\mathbf{c}, \mathbf{i}, n_0)$, where $\mathbf{c}$ is a vector containing nonzero elements of $\mathbf{x}$, $\mathbf{i}$ is an integer vector containing the indices of elements stored in $\mathbf{c}$ and n_0 specifies the number of nonzero elements. The stored elements are related to the original vector by formula $x_{\{i_j\}} = c_j$ for $j = 1, \ldots, n_0$. To give an example, the vector $\mathbf{x} = (0, 0, 3, 0, -8, 1.5, 0, 0, 0, 16, 0)$ could be stored as

$$\mathbf{c} = (3, 1.5, -8, 16), \quad \mathbf{i} = (3, 6, 5, 10), \quad n_0 = 4.$$

Obviously, there is no need to store the elements in the original order. Therefore, adding new nonzero elements is easy. Operations involving more sparse vectors are simpler if we can directly access elements of one vector, that is, if one of the vectors is "uncompressed." For example, computing the inner product $a = \mathbf{x}^\top \mathbf{y}$ of a sparse vector $\mathbf{x}$ stored in the compressed form with a sparse uncompressed vector $\mathbf{y}$ follows the algorithm

$$a = 0; \qquad \text{for } j = 1, \ldots, n_0 : \quad a = a + y_{\{i_j\}} \cdot c_j.$$

The compressed row storage for matrices is a generalization of the vector concept. We store the nonzero elements of $\mathbf{A}$ as a set of sparse row (or column) vectors in the compressed form. The main difference is that, instead of a single number n_0, we need to store a whole vector $\mathbf{n_0}$ specifying the positions of the first row elements of $\mathbf{A}$ in $\mathbf{c}$. For example, the matrix

$$\mathbf{A} = \begin{pmatrix} A_{11} & A_{12} & 0 & 0 & 0 & 0 \\ A_{21} & 0 & 0 & A_{24} & 0 & 0 \\ 0 & 0 & 0 & A_{34} & 0 & 0 \\ 0 & 0 & A_{43} & 0 & A_{45} & 0 \\ 0 & A_{52} & 0 & 0 & 0 & A_{56} \end{pmatrix}$$

would be represented rowwise as

$$\begin{aligned} \mathbf{c} &= (A_{11}, A_{12} | A_{21}, A_{24} | A_{34} | A_{43}, A_{45} | A_{52}, A_{56}), \\ \mathbf{i} &= (1, 2 | 1, 4 | 4 | 3, 5 | 2, 6), \\ \mathbf{n_0} &= (1, 3, 5, 6, 8, 10). \end{aligned}$$

(The sign "|" just emphasizes the end of a row and has no consequence for the storage itself.) As in the case of vectors, the elements in each row do not have to be ordered. Consequently, there is no direct access to a particular element A_{ij} stored in $\mathbf{c}$. Nevertheless, retrieving a row is easy: it suffices to examine the part of $\mathbf{i}$ corresponding to the ith row, which is given by $\mathbf{n_0}$. On the contrary, retrieving a column involves a search through the whole storage scheme. Therefore, if a fast access to columns is necessary, it is preferable to simultaneously store $\mathbf{A}$ rowwise and columnwise.

A special type of sparse matrices are matrices with a banded structure.

Definition 2. *The row bandwidth of a matrix* $\mathbf{A} \in \mathbb{R}^{m\times n}$ *is defined as*

$$w(\mathbf{A}) = \max_{1\le i\le m} (l_i(\mathbf{A}) - f_i(\mathbf{A}) + 1),$$

where $f_i(\mathbf{A}) = \min\{j|\mathbf{A}_{ij} \neq 0\}$ *and* $l_i(\mathbf{A}) = \max\{j|\mathbf{A}_{ij} \neq 0\}$ *are column indices of the first and last nonzero elements in the* i*th row of* $\mathbf{A}$*.*

A banded matrix $\mathbf{A}$ is considered to be sparse if $w(\mathbf{A}) \ll n$. Contrary to the general case, vector $\mathbf{c}$ of a banded matrix typically contains for each row all elements between the first and last nonzero ones. Thus, the storage scheme does not have to include in $\mathbf{i}$ all column indices, only one index for the first nonzero element in a row. On the other hand, zeros within the band have to be stored as well. For example, the matrix

$$\mathbf{A} = \begin{pmatrix} A_{11} & A_{12} & 0 & 0 & 0 \\ 0 & A_{22} & 0 & A_{24} & 0 \\ 0 & 0 & 0 & A_{34} & 0 \\ 0 & 0 & A_{43} & 0 & A_{45} \end{pmatrix}$$

would be represented as

$$\begin{aligned} \mathbf{c} &= (A_{11}, A_{12}|A_{22}, 0, A_{24}|A_{34}|A_{43}, 0, A_{45}), \\ \mathbf{i} &= (1, 2, 4, 3), \\ \mathbf{n}_0 &= (1, 3, 6, 7, 10). \end{aligned}$$

An interesting observation is that the row bandwidth $w(\mathbf{A})$ can be influenced by column permutations. The fill-minimizing column orderings are discussed by Björck (1996) and George and Ng (1983), for instance.

Details on some other storage schemes can be found in Björck (1996), Duff et al. (1989) and Press et al. (1992).

5.2 Methods for Sparse Matrices

Methods for sparse matrices are still subject to intensive research. Moreover, the choice of a suitable method for a given problem (and even the choice of an algorithm for elementary operations such as matrix-vector multiplication) depends on many factors, including dimension, matrix type storage scheme, and computational environment (e.g., storage in virtual memory vs. auxiliary storage; vector vs. parallel computing, etc.). Therefore, we provide only a general overview and references to most general results. More details can be found in Björck (1996), Dongarra and Eijkhout (2000), Duff et al. (1989), Hackbusch (1994) and Saad (2003).

First, many discussed algorithms can be relatively easily adopted for banded matrices. For example, having a row-based storage scheme, one just needs to modify the summation limits in the row version of Cholesky decomposition. Moreover, the positions of nonzero elements can be determined in advance (Ng and Peyton, 1993).

Second, the algorithms for general sparse matrices are more complicated. A graph representation may be used to capture the nonzero pattern of a matrix as well as to predict the pattern of the result (e.g., the nonzero pattern of $\mathbf{A}^\top\mathbf{A}$, the Cholesky factor $\mathbf{U}$, etc.). To give an overview, methods adopted for sparse matrices include, but are not limited to, usually used decompositions (e.g., Cholesky, Ng and Peyton, 1993; LU and LDU, Mittal and Al-Kurdi, 2002; QR, George and Liu, 1987, and Heath, 1984), solving systems of equations by direct (Gupta, 2002; Tran et al., 1996) and iterative methods (Makinson and Shah, 1986; Zlatev and Nielsen, 1988) and searching eigenvalues (Bergamaschi and Putti, 2002; Golub et al., 2000).

References

Allanelli, M. and Hadjidimos, A. (2004). Block Gauss elimination followed by a classical iterative method for the solution of linear system. *Journal of Computational and Applied Mathematics*: in press.

Anderson, E., Bai, Z., Bischof, C., Blackford, S., Demmel, J., Dongarra, J., Croz, J.D., Greenbaum, A., Hammarling, S., McKenney, A. and Sorensen, D. (1999). *LAPACK Users' Guide, Third Edition*. SIAM Press, Philadelphia, USA.

Axelsson, O. (1994). *Iterative Solution Methods*. Cambridge University Press, Cambridge, UK.

Bergamaschi, L. and Putti, M. (2002). Numerical comparison of iterative eigensolvers for large sparse symmetric positive definite matrices. *Computer Methods in Applied Mechanics and Engineering*, 191: 5233–5247.

Benoit, C. (1924). Note sure une methode de resolution des equation mormales provenant de l'application de la methode des moindres carres a un systeme e'equations lineaires en nombre inferieure a celuides inconnues. Application de la methode a la resolution d'un systeme defini d'equations lineaires (Procede du Commandant Cholesky). *Bulletin geodesique*, 24/2: 5–77.

Björck, A. (1994). Numerics of Gram-Schmidt Orthogonalization. *Linear Algebra and Its Applications*, 198: 297–316.

Björck, A. (1996). *Numerical Methods for Least Squares Problems*. SIAM Press, Philadelphia, USA.

Croz, J.D. and Higham, N.J. (1992). Stability of methods for matrix inversion. *IMA Journal of Numerical Analysis*, 12: 1–19.

Dax, A. (2000). A modified Gram-Schmidt algorithm with iterative orthogonalization and column pivoting. *Linear Algebra and Its Applications*, 310: 25–42.

Demmel, J.W., Gu, M., Eisenstat, S., Slapničar, I., Veselić, K. and Drmač, Z. (1999). Computing the singular value decomposition with high relative accuracy. *Linear Algebra and its Applications*, 299: 21–80.

Dongarra, J.J. and Eijkhout, V. (2000). Numerical linear algebra algorithms and software. *Journal of Computational and Applied Mathematics*, 123: 489–514.

Duff, I.S., Erisman, A.M. and Reid, J.K. (1989). *Direct Methods for Sparse Matrices*. Oxford University Press, USA.

Freund, R. and Nachtigal, N. (1991). QMR: a quasi-minimal residual method for non-Hermitian linear systems. *Numerical Mathematics*, 60: 315–339.

Gallivan, K.A., Plemmons, R.J. and Sameh, A.H. (1990). Parallel algorithms for dense linear algebra computations. *SIAM Review*, 32: 54–135.

Gentle, J.E. (1998). *Numerical Linear Algebra for Applications in Statistics*. Springer, New York, USA.

Gentleman, W.M. (1973). Least squares computations by Givens transformations without square roots. *Journal of Institute of Mathematics and its Applications*, 12: 329–336,

Gentleman, W.M. (1975). Error analysis of QR decomposition by Givens transformations. *Linear Algebra and its Applications*, 10: 189–197.

George, A. and Liu, J.W.H. (1987). Householder reflections versus givens rotations in sparse orthogonal decomposition. *Linear Algebra and its Applications*, 88: 223–238.

George, J.A. and Ng, E.G. (1983). On row and column orderings for sparse least squares problems. *SIAM Journal of Numerical Analysis*, 20: 326–344.

Givens, W. (1958). Computation of Plane Unitary Rotations Transforming a General Matrix to Triangular Form. *Journal of SIAM*, 6/1: 26–50.

Golub, G.H. (1965). Numerical methods for solving least squares problems. *Numerical Mathematics*, 7: 206–216.

Golub, G.H. and Kahan, W. (1965). Calculating the singular values and pseudo-inverse of a matrix. *SIAM Journal on Numerical Analysis B*, 2: 205–224.

Golub, G.H. and Reinsch, C. (1970). Singular value decomposition and least squares solution. *Numerical Mathematics*, 14: 403–420.

Golub, G.H. and van Loan, C.F. (1996). *Matrix Computations*. Johns Hopkins University Press, Baltimore, Maryland.

Golub, G.H., Zhang, Z. and Zha, H. (2000). Large sparse symmetric eigenvalue problems with homogeneous linear constraints: the Lanczos process with inner-outer iterations. *Linear Algebra and its Applications*, 309: 289–306.

Gupta, A. (2002). Recent Advances in Direct Methods for Solving Unsymmetric Sparse Systems of Linear Equations. *ACM Transactions on Mathematical Software*, 28: 301–324.

Hackbusch, W. (1994). *Iterative Solution of Large Sparse Systems of Equations*. Springer, New York, USA.

Hadjidimos, A. (2000). Successive Overrelaxation (SOR) and related methods. *Journal of Computational and Applied Mathematics*, 123: 177–199.

Hammarling, S. (1974). A note on modifications to the Givens plane rotation. *Journal of Institute of Mathematics and its Applications*, 13: 215–218.

Hari, V. and Veselić, K. (1987). On Jacobi methods for singular value decompositions. *SIAM Journal of Scientific and Statistical Computing*, 8: 741–754.

Harville, D.A. (1997). *Matrix Algebra from a Statistician's Perspective*. Springer, New York, USA.

Heath, M.T. (1984). Numerical methods for large sparse linear least squares problems. *SIAM Journal of Scientific and Statistical Computing*, 26: 497–513.

Hestenes, M.R. and Stiefel, E. (1952). Method of conjugate gradients for solving linear systems. *J. Res. Nat Bur. Standards B*, 49: 409–436.

Higham, N.J. (1989). The accuracy of solutions to triangular systems. *SIAM Journal on Numerical Analysis*, 26: 1252–1265.

Higham, N.J. (1997). Recent Developments in Dense Numerical Linear Algebra. In Duff, I.S. and Watson, G.A. (eds), *State of the Art in Numerical Analysis*, Oxford University Press, Oxford.

Higham, N.J. (2000). QR factorization with complete pivoting and accurate computation of the SVD. *Linear Algebra and its Applications*, 309: 153–174.

Higham, N.J. (2002). *Accuracy and Stability of Numerical Algorithms*, Second edition. SIAM Press, Philadelphia, USA.

Hong, Y.P. and Tan, C.T. (1992). Rank-revealing QR fatorizations and the singular value decomposition. *Mathematics of Computation*, 58: 213–232.

Householder, A.S. (1958). Unitary triangularization of a nonsymmetric matrix. *Journal of the Association of Computing Machinery*, 5: 339–342.

Ipsen, I.C.F. (1997). Computing an Eigenvector with Inverse Iteration. *SIAM Review*, 39: 254–291.

Kahan, W. (1958). *Gauss-Seidel methods of solving large systems of linear equations*. Doctoral thesis, University of Toronto, Toronto, Canada.

Kammerer, W.J. and Nashed, M.Z. (1972). On the convergence of the conjugate gradient method for singular linear operator equations. *SIAM Journal on Numerical Analysis*, 9: 165–181.

Makinson, G.J. and Shah, A.A. (1986). An iterative solution method for solving sparse nonsymmetric linear systems. *Journal of Computational and Applied Mathematics*, 15: 339–352.

Martin, R.S., Peters, G. and Wilkinson, J.H. (1965). Symmetric decomposition of a positive definite matrix. In Wilkinson, J.H. and Reinsch, C. (eds), *Linear Algebra (Handbook for Automation Computation, Vol. II)*. Springer, Heidelberg, Germany.

Meinguet, J. (1983). Refined error analysis of cholesky factorization. *SIAM Journal on Numerical Analysis*, 20: 1243–1250.

Milaszewicz, J.P. (1987). Improving Jacobi and Gauss-Seidel Iterations. *Linear Algebra and Its Applications*, 93: 161–170.

Miranian, L. and Gu, M. (2003). Strong rank revealing LU factorizations. *Linear Algebra and its Applications*, 367: 1–16.

Mittal, R.C. and Al-Kurdi, A. (2002). LU-decomposition and numerical structure for solving large sparse nonsymmetric linear systems. *Computers & Mathematics with Applications*, 43: 131–155.

Ng, E.G. and Peyton, B.W. (1993). Block Sparse Cholesky Algorithm on Advanced Uniprocessor Computers. *SIAM Journal of Scientific Computing*, 14: 1034–1056.

Nool, M. (1995). Explicit parallel block Cholesky algorithms on the CRAY APP. *Applied Numerical Mathematics*, 19: 91-114.

Pan, C.T. (2000). On the existence and computation of rank revealing LU factorizations. *Linear Algebra and its Applications*, 316: 199–222.

Press, W.H., Teukolsky, S.A., Vetterling, W.T. and Flannery, B.P. (1992). *Numerical Recipes in C: the Art of Scientific Computing, Second Edition.* Cambridge University Press, Cambridge, UK.

Rice, J.R. (1966). Experiments on Gram-Schmidt orthogonalization. *Mathematics of Computation*, 20: 325–328.

Saad, Y. (2003). *Iterative Methods for Sparse Linear Systems. Second Edition.* SIAM Press, USA.

Saad, Y. and Schultz, M.H. (1986). GMRES: a generalized minimal residual algorithm for solving nonsymmetric linear systems. *SIAM Journal of Scientific and Statistical Computing*, 7: 856–869.

Sidi, A. (1989). On extensions of the power method for normal operators. *Linear Algebra and Its Applications*, 120: 207–224.

Skeel, R.D. (1980). Iterative refinement implies numerical stability for Gaussian elimination. *Mathematics of Computation*, 35: 817–832.

Stewart, G.W. (1976). The economical storage of plane rotations. *Numerical Mathematics*, 25: 137–138.

Stewart, G.W. (1998). *Matrix Algorithms, Volume I: Basic Decompositions.* SIAM Press, Philadelphia, USA.

Stoer, J. and Bulirsch, R. (2002). *Introduction to Numerical Analysis, Third Edition.* Springer, New York, USA.

Tran, T.M., Gruber, R., Appert, K. and Wuthrich, S. (1996). A direct parallel sparse matrix solver. *Computer Physics Communications*, 96: 118–128.

Trefethen, L.N. and Bau, D. (1997). *Numerical Linear Algebra.* SIAM Press, Philadelphia, USA.

von Matt, U. (1995). The Orthogonal QD-Algorithm. In Moonen, M. and De Moor, B. (eds), *SVD and Signal Processing, III: Algorithms, Architectures and Applications*, Elsevier, Amsterdam.

Vorst, V.D. (1992). Bi-CGSTAB: A fast and smoothly converging variant of Bi-CG for the solution of nonsymmetric linear systems. *SIAM Journal of Scientific and Statistical Computing*, 13: 631–644.

Wampler, R.H. (1970). A report on the accuracy of some widely used least squares computer programs. *Journal of American Statistical Association*, 65: 549–565.

Young, D.M. (1954). Iterative methods for solving partial differential equations of elliptic type. *Transactions of the American Mathematical Society*, 76: 92–111.

Zhou, B.B. and Brent, R.P. (2003). An efficient method for computing eigenvalues of a real normal matrix. *Journal of Parallel and Distributed Computing*, 63: 638–648.

Zlatev, Z. and Nielsen, H.B. (1988). Solving large and sparse linear least-squares problems by conjugate gradient algorithms. *Computers & Mathematics with Applications*, 15: 185–202.

Zou, Q. (1991). An observation on Gauss elimination. *Computers and Mathematical Applications*, 22: 69–70.

Index